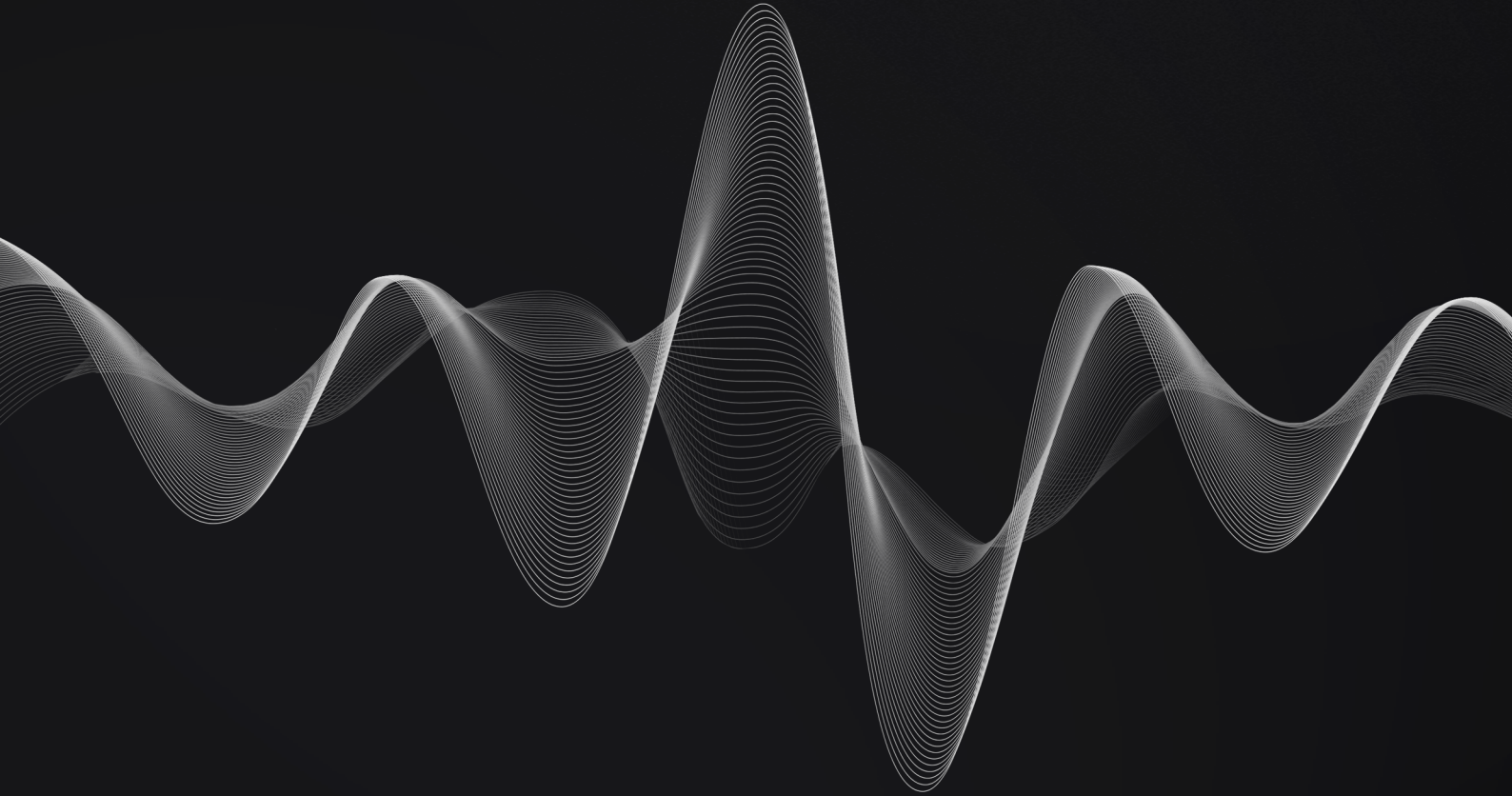




# FOMO

## FomoToken Smart Contract Audit Report






# Document Control

**PUBLIC**

**FINAL**(v2.1)

## Audit\_Report\_FOMO-FOM\_FINAL\_21

Dec 18, 2024		v0.1	João Simões: Initial draft
Dec 18, 2024		v0.2	João Simões: Added findings
Dec 18, 2024		v1.0	Charles Dray: Approved
Jan 22, 2025		v1.1	João Simões: Reviewed findings
Jan 22, 2025		v2.0	Charles Dray: Finalized
Jan 22, 2025		v2.1	Charles Dray: Published

<b>Points of Contact</b>	Apeinfomo	Fomo	dev@apeinfomo.app
	Charles Dray	Resonance	charles@resonance.security
<b>Testing Team</b>	João Simões	Resonance	joao@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security
	Luis Arroyo	Resonance	luis.arroyo@resonance.security

## Copyright and Disclaimer

© 2024 Resonance Security, Inc. All rights reserved.

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

# Contents

<b>1 Document Control</b>	<b>2</b>
Copyright and Disclaimer .....	2
<b>2 Executive Summary</b>	<b>4</b>
System Overview .....	4
Repository Coverage and Quality.....	4
<b>3 Target</b>	<b>5</b>
<b>4 Methodology</b>	<b>6</b>
Severity Rating.....	7
Repository Coverage and Quality Rating.....	8
<b>5 Findings</b>	<b>9</b>
Floating Pragma .....	10
Unused Standard Libraries.....	11
<b>A Proof of Concepts</b>	<b>12</b>

# Executive Summary

**Fomo** contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between December 17, 2024 and December 19, 2024. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 2 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Fomo with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.



## System Overview

FomoToken is a simple ERC20 token deployed on Sepolia testnet. An initial supply of 1 billion tokens is minted to the deployer of the contract and all usual functionalities of ERC20 tokens are available. Additionally, it implements a burning mechanism through the ERC20Burnable extension and gasless approves via the ERC20Permit extension.



## Repository Coverage and Quality

*This section of the report has been redacted by the request of the customer.*

# Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Link (on Dec 17th, 2024): [FomoToken.sol](#) at [Sepolia](#)

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

# Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

## Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions



## Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info



# Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.



# Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance’s Testing Team. This metric characterizes findings as follows:

- ..... **"Quick Win"** Requires little work for a high impact on risk reduction.
- ....|.. **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ...||| **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

---

RES-01	Floating Pragma	.....	Acknowledged
RES-02	Unused Standard Libraries	.....	Acknowledged



# Floating Pragma

Info

RES-FOMO-FOM01

Code Quality

Acknowledged

## Code Section

- [FomoToken.sol#F1#L3](#)

## Description

Floating pragmas is a feature of Solidity that allows developers to specify a range of compiler versions that can be used to compile the smart contract code. For security purposes specifically, the usage of floating pragmas is discouraged and not recommended. Contracts should be compiled and deployed with a strict pragma, the one which was thoroughly tested the most by developers. Locking the pragma helps ensure that contracts do not accidentally get deployed using too old or too recent compiler versions that might introduce security issues that impact the contract negatively.

It should be noted however that, pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package.

The smart contract of FomoToken makes use of a floating pragma. This is not intended for use as a library for other developers and, as such, should be locked of its pragma.

## Recommendation

It is recommended to lock the pragma of all smart contracts to the same Solidity compiler version.

## Status

*The issue was acknowledged by FOMO's team. The development team stated "Due to being such a simple token, no modifications will be made at this time."*



# Unused Standard Libraries

Info

RES-FOMO-FOM02

Code Quality

Acknowledged

## Code Section

- Not specified.

## Description

OpenZeppelin implements several standards used all across blockchain development, especially in Solidity and the Ethereum Virtual Machine. These standards primary goal is to unify and normalize development patterns so that the entire blockchain may be more understandable and readily usable.

The FomoToken smart contract implements source code that mimics or is very similar to components already implemented by OpenZepellin, and could therefore, be switched with the more the standard well-known approach. Such source code includes:

- `Counters` contract, can be substituted with OpenZeppelin's implementation of `Nonces`.
- `ERC20Permit` functionalities, can be substituted with OpenZeppelin's implementation of `ERC20Permit`.

It should be noted that the usage of bigger libraries does not constitute an overuse of gas, since only the functions that are used are included on the bytecode of the smart contract.

Some libraries were also identified to be out-of-date.

## Recommendation

It is recommended to make use of implemented and audited standards that already solve the necessary functionalities, and maintain these up-to-date.

## Status

*The issue was acknowledged by FOMO's team. The development team stated "Due to being such a simple token, no modifications will be made at this time."*

# Proof of Concepts

*No Proof-of-Concept was deemed relevant to describe findings in this engagement.*